MEET VUE JS

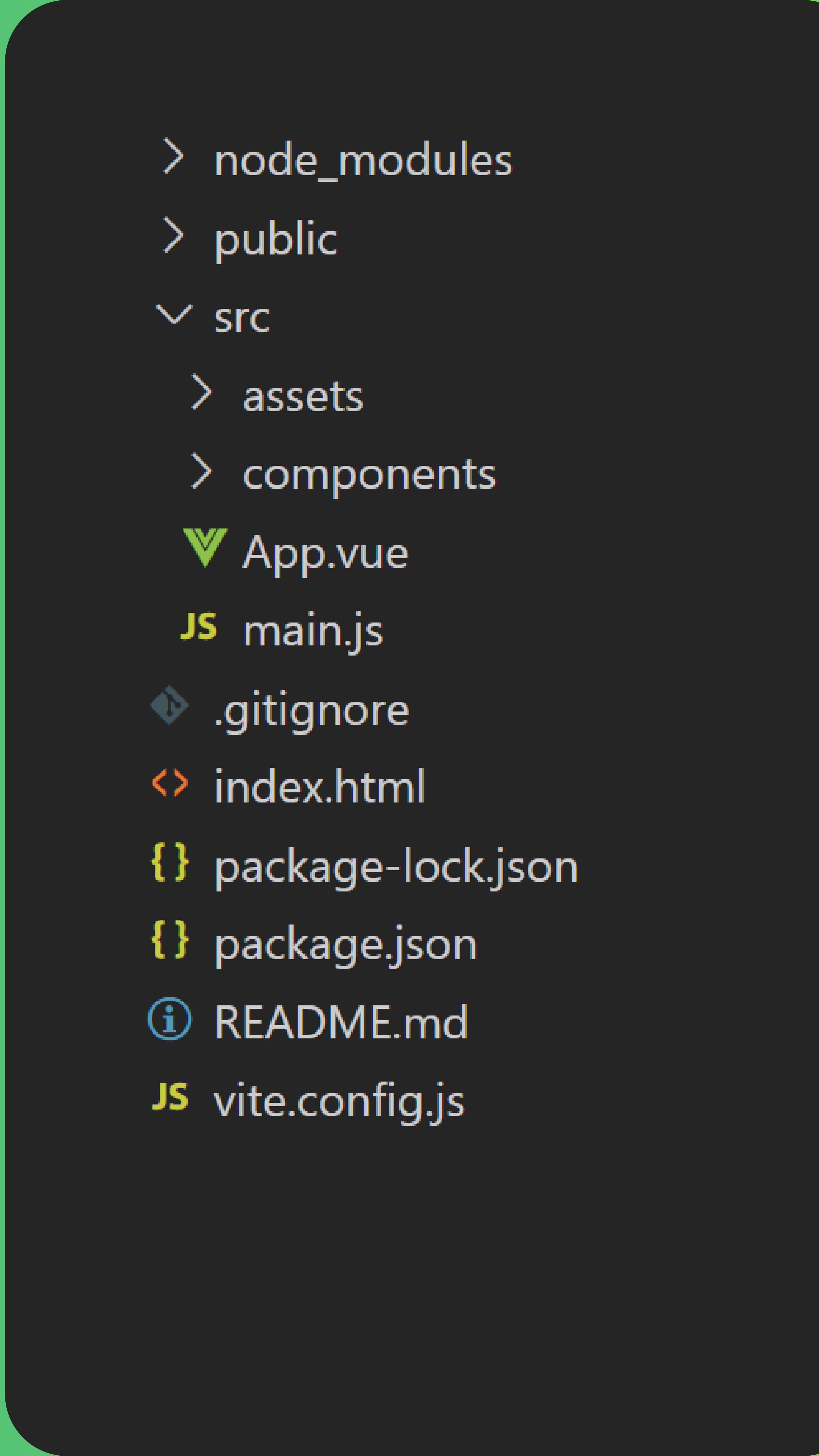
- Popular JavaScript front-end framework.
- Simplifies UI development with reusable components
- Efficient data binding for fast and responsive updates.
- Small learning curve and can be easily integrated with other libraries.
- Large and active community of users and contributors.

CREATE NEW PROJECT

```
npm init vue@latest
```

```
npm create vite@latest
```

```
"scripts": {
  "dev": "vite",
  "build": "vite build",
  "preview": "vite preview"
},
```



```
> node_modules
> public
v src
  > assets
  > components
  App.vue
  main.js
.gitignore
index.html
package-lock.json
package.json
README.md
vite.config.js
```

Vue Component

Script Setup

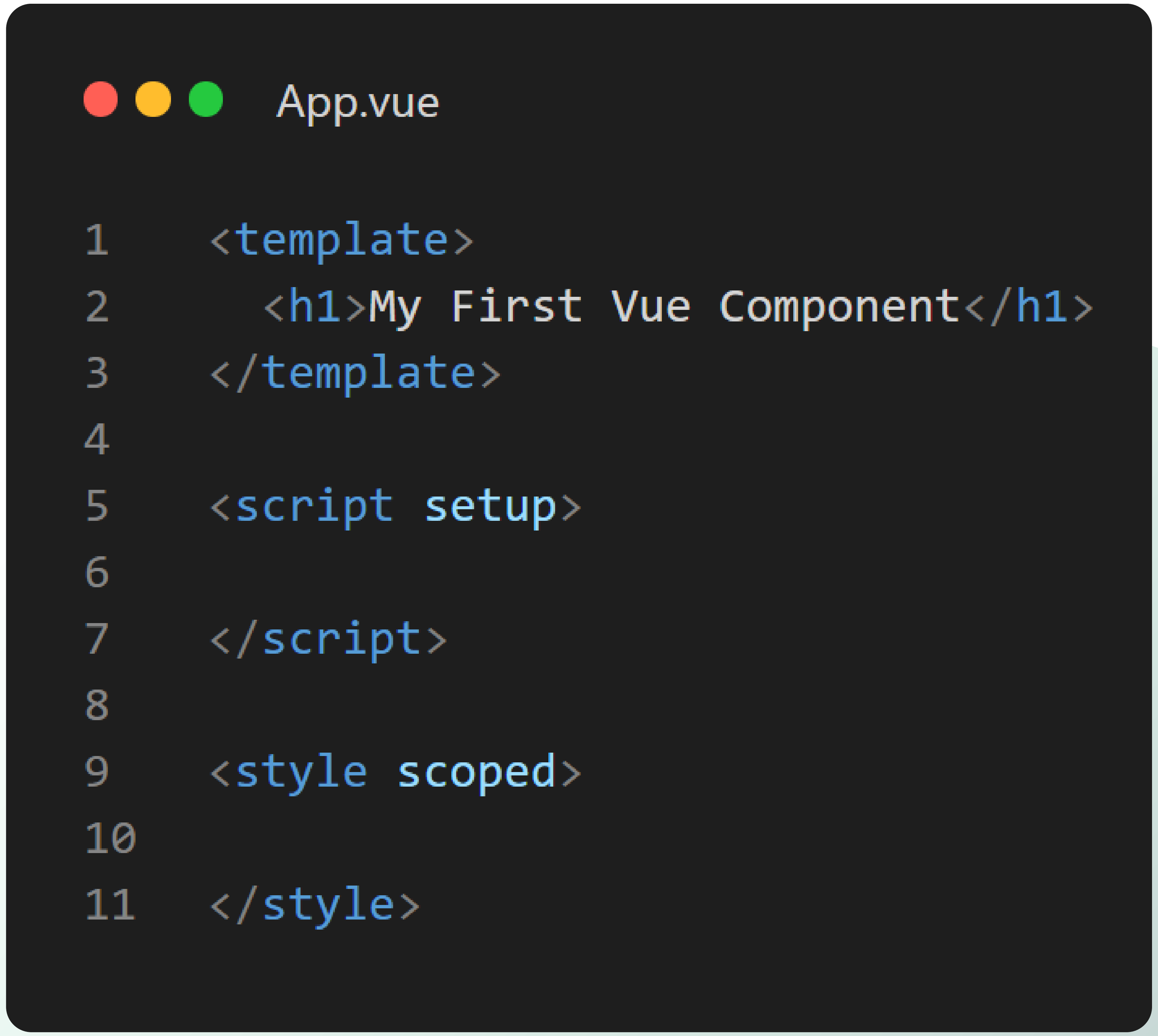
- When using variables, function declarations, and imports
- When working with reactive state
- For importing component, dynamic component
- And many more, we will discover through our learning

Style Scoped

- When a <style> tag has the scoped attribute, its CSS will apply to elements of the current component only.

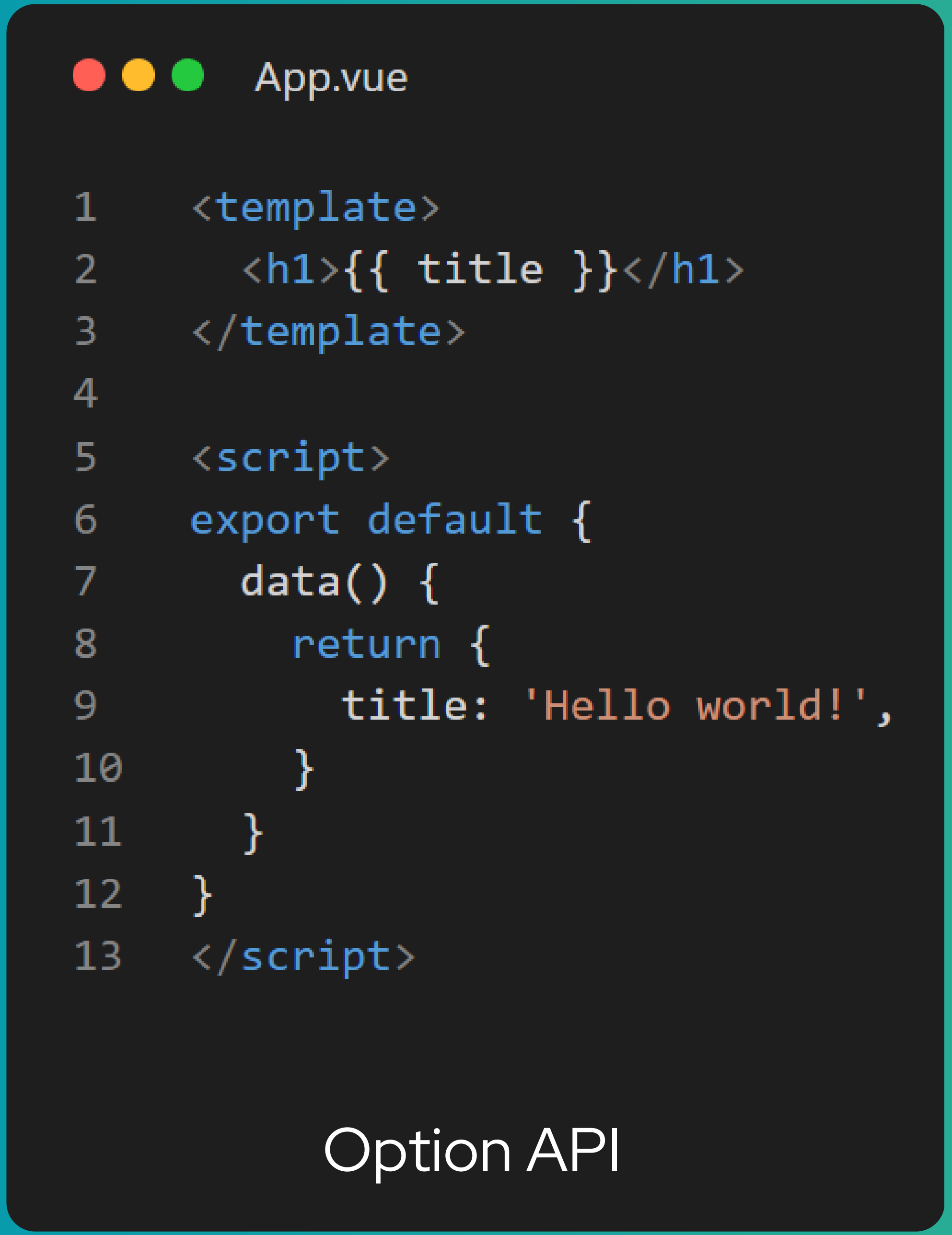
Template

- The <template> tag is used as a placeholder when we want to use a built-in directive without rendering an element in the DOM


App.vue

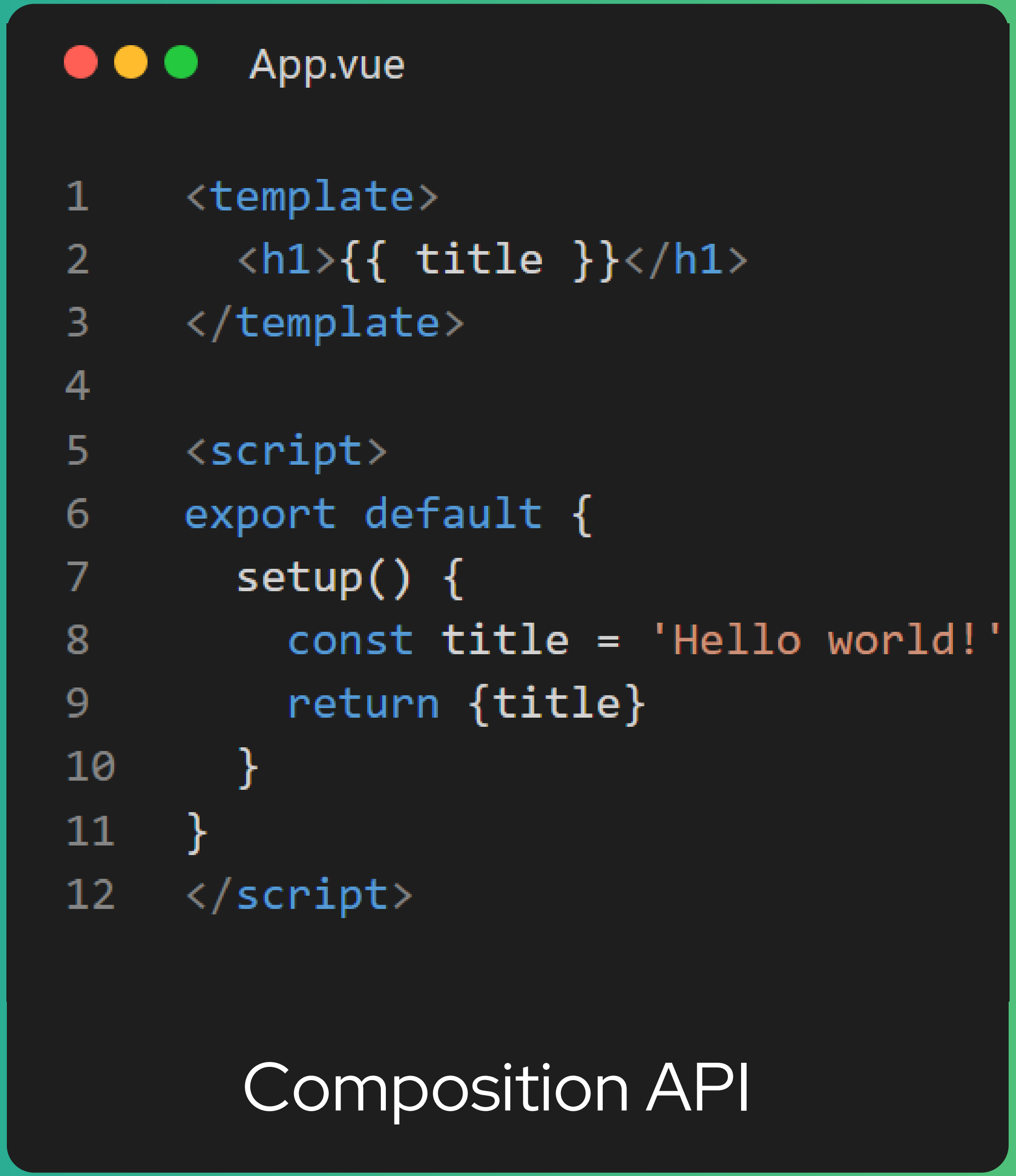
```
<template>
  <h1>My First Vue Component</h1>
</template>

<script setup>

</script>

<style scoped>

</style>
```

WAYS TO COMPOSE COMPONENTS



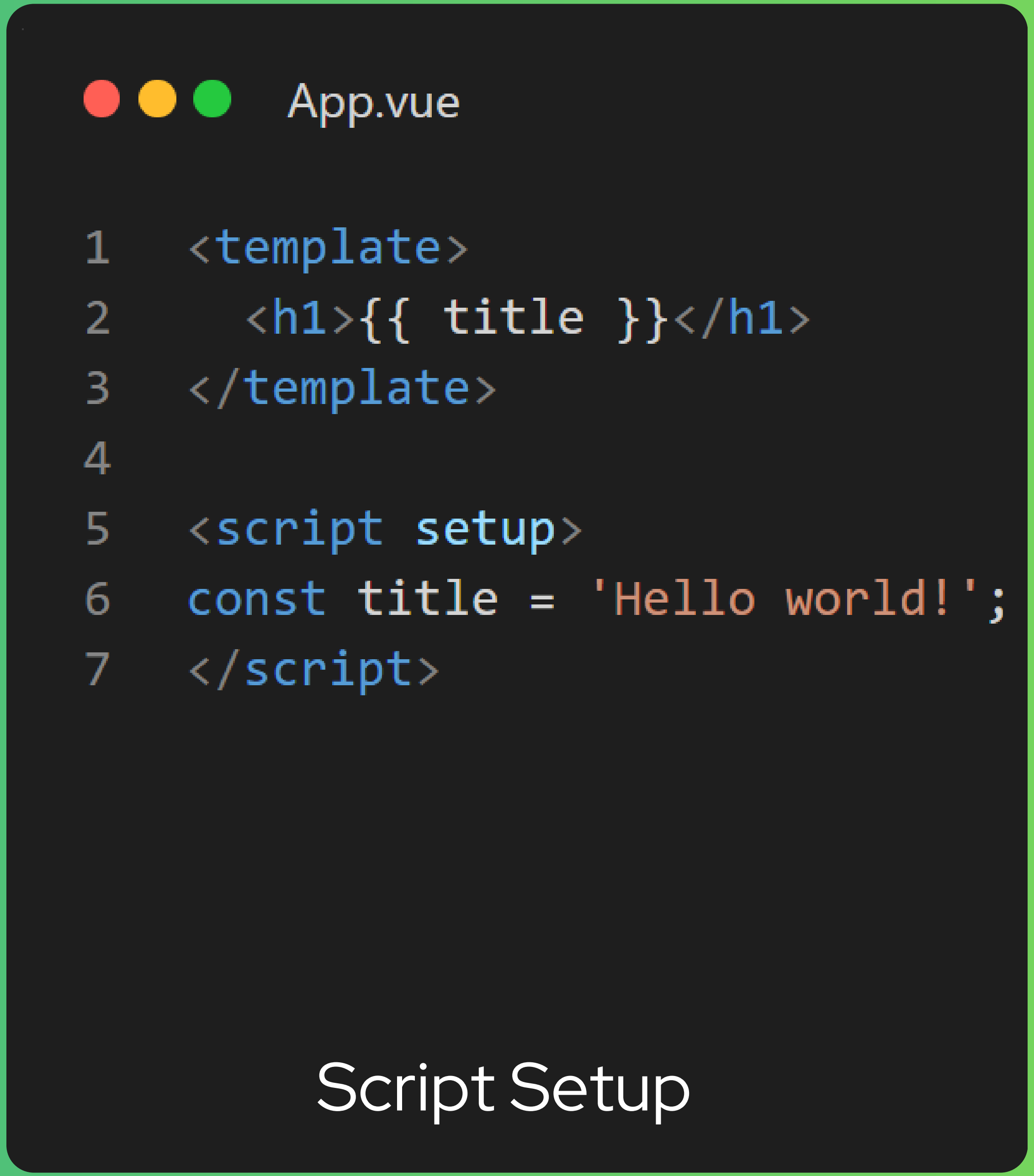
```
<template>
  <h1>{{ title }}</h1>
</template>

<script>
export default {
  data() {
    return {
      title: 'Hello world!',
    }
  }
}
</script>
```

Option API



```
<template>
  <h1>{{ title }}</h1>
</template>

<script>
export default {
  setup() {
    const title = 'Hello world!'
    return {title}
  }
}
</script>
```

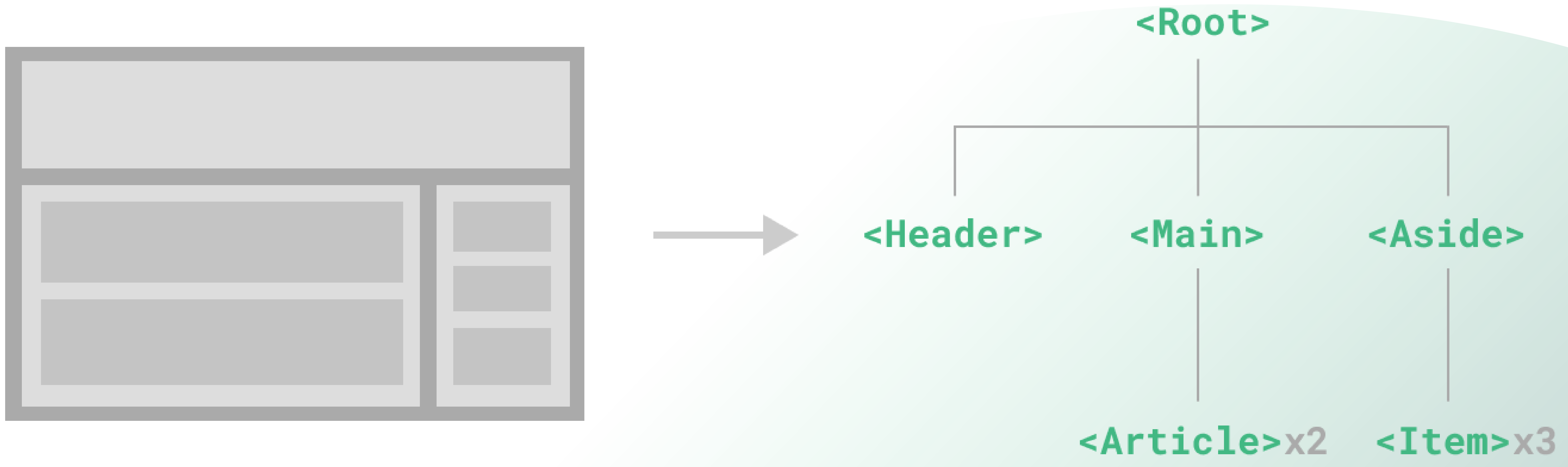
Composition API



```
<template>
  <h1>{{ title }}</h1>
</template>

<script setup>
const title = 'Hello world!';
</script>
```

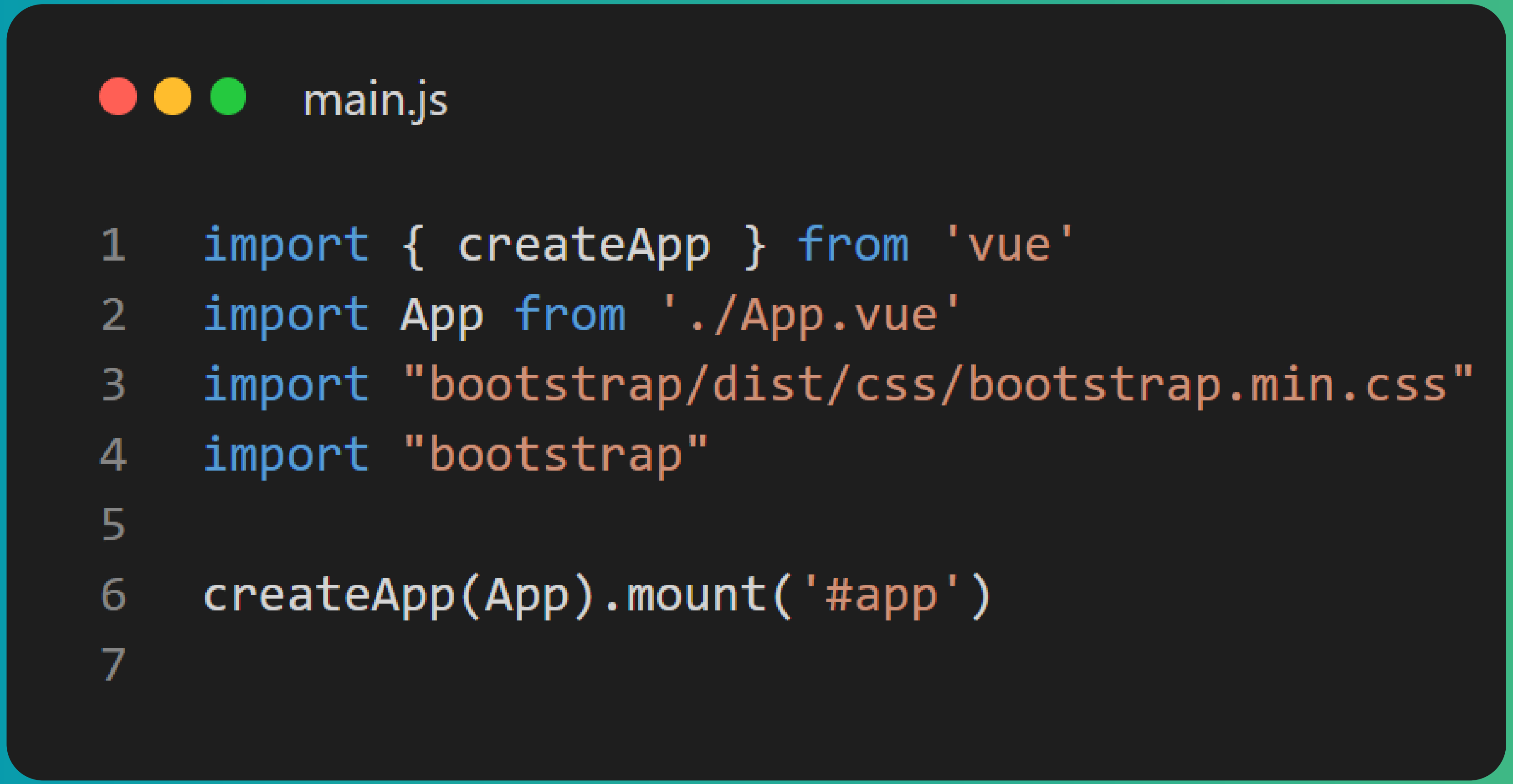
Script Setup

Component Tree Concept



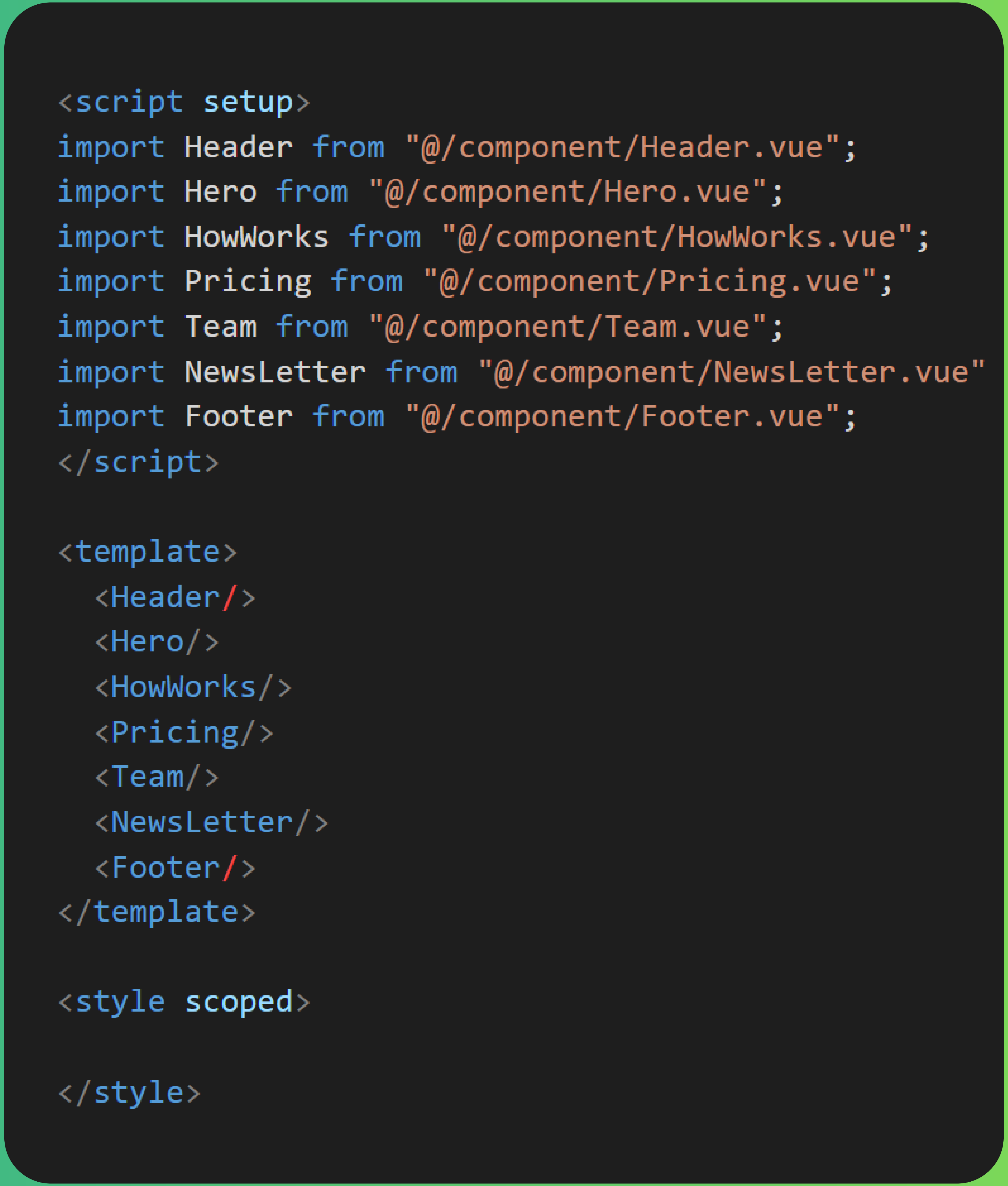
MAKE ONE PAGE TEMPLATE

Using bootstrap



main.js

```javascript
import { createApp } from 'vue'
import App from './App.vue'
import "bootstrap/dist/css/bootstrap.min.css"
import "bootstrap"

createApp(App).mount('#app')
```



```vue
<script setup>
import Header from "@/component/Header.vue";
import Hero from "@/component/Hero.vue";
import HowWorks from "@/component/HowWorks.vue";
import Pricing from "@/component/Pricing.vue";
import Team from "@/component/Team.vue";
import NewsLetter from "@/component/NewsLetter.vue"
import Footer from "@/component/Footer.vue";
</script>

<template>
  <Header/>
  <Hero/>
  <HowWorks/>
  <Pricing/>
  <Team/>
  <NewsLetter/>
  <Footer/>
</template>

<style scoped>

</style>
```

TEMPLATE SYNTAX

Text Interpolation

The most basic form of data binding is text interpolation using the "Mustache" syntax (double curly braces)

```
<template>
  <h1>Message: {{ msg }}</h1>
  <h1>Message: <span v-text="msg"></span></h1>
</template>

<script setup>
  let msg="Hello"
</script>
```

TEMPLATE SYNTAX

Raw HTML

The double mustaches interpret the data as plain text, not HTML. In order to output real HTML, you will need to use the v-html directive

```
<template>
  <p>Using text interpolation: {{ rawHtml }}</p>
  <p>Using v-html directive: <span v-html="rawHtml"></span></p>
</template>

<script setup>
  let rawHtml="<button>Hello</button>"
</script>
```

TEMPLATE SYNTAX

Attribute Bindings

Mustaches cannot be used inside HTML attributes. Instead, use a v-bind directive

```
<template>
  <div v-bind:id="dynamicId">Hello</div>
  <img :src="imgSrc"/>
  <button :disabled="isButtonDisabled">Button</button>
</template>

<script setup>
  let dynamicId="myId"
  let imgSrc="https://cdn.rabbil.com/photos/images/2022/11/04/rabbilVai.png"
  let isButtonDisabled=true
</script>
```

TEMPLATE SYNTAX

Binding Multiple Attributes

```
<template>
  <button v-bind="objectOfAttrs">Button</button>
</template>

<script setup>
  let objectOfAttrs ={id:'myBtn',class: 'btn btn-primary'}
</script>
```

TEMPLATE SYNTAX

Using JavaScript Expressions

- Inside text interpolations (mustaches)
- In the attribute value of any Vue directives (special attributes that start with v-)

```
<template>
  <p>{{number + 1 }}</p>
  <p>{{ok ? 'YES' : 'NO' }}</p>
  <p>{{message.split('').reverse().join('') }}</p>
  <button :id="`list-${id}`">Button</button>
</template>

<script setup>
  let number =1;
  let ok=true;
  let message="ABCD";
  let id='myID'
</script>
```

V-TEXT DIRECTIVES

Update the element's text content.

```
<template>
  <span v-text="msg"></span>
  <!-- same as -->
  <span>{{msg}}</span>
</template>

<script setup>
const msg = "Hello";
</script>
```

V-HTML DIRECTIVES

Update the element's innerHTML

```
<template>
  <span v-html="btn"></span>
</template>

<script setup>
const btn = "<button>Button</button>";
</script>
```

V-SHOW DIRECTIVES

Toggle the element's visibility based on the truthy-ness of the expression value.

```
<template>
  <div v-show="message" id="app">
    Hello I'm visible.
  </div>
</template>

<script setup>
let message=false;
</script>
```

V-IF DIRECTIVES

Conditionally render an element or a template fragment based on the truthy-ness of the expression value.

```
<template>
  <h2 v-if="data>50">
    data is greater than 50
  </h2>
  <h2 v-else-if="data<50">
    data is smaller than 50
  </h2>
  <h2 v-else>
    data is equal to 50
  </h2>
</template>

<script setup>
const data = 100;
</script>
```

V-FOR DIRECTIVES

Render the element or template block multiple times based on the source data.

```
<template>
  <ul>
    <li v-for="item in list">
      {{item}}
    </li>
  </ul>
</template>

<script setup>
const list = ['Afghanistan', 'Albania', 'Algeria']
</script>
```

V-ON SHORTHAND @ DIRECTIVES

Attach an event listener to the element.

```
<template>
  <button @click="doThis">Click</button>
</template>

<script setup>
function doThis(){
  alert('do this')
}
</script>
```

V-ON SHORTHAND @ DIRECTIVES

Attach an event listener to the element.

```
<!-- method handler -->
<button v-on:click="doThis"></button>

<!-- dynamic event -->
<button v-on:[event]="doThis"></button>

<!-- inline statement -->
<button v-on:click="doThat('hello', $event)"></button>

<!-- shorthand -->
<button @click="doThis"></button>

<!-- shorthand dynamic event -->
<button @[event]="doThis"></button>

<!-- stop propagation -->
<button @click.stop="doThis"></button>
```

V-ON SHORTHAND @ DIRECTIVES

Attach an event listener to the element.

```html
<!-- prevent default -->
<button @click.prevent="doThis"></button>

<!-- prevent default without expression -->
<form @submit.prevent></form>

<!-- chain modifiers -->
<button @click.stop.prevent="doThis"></button>

<!-- key modifier using keyAlias -->
<input @keyup.enter="onEnter" />

<!-- the click event will be triggered at most once -->
<button v-on:click.once="doThis"></button>

<!-- object syntax -->
<button v-on="{ mousedown: doThis, mouseup: doThat }"></button>
```

PROPS PASSING

Passing properties from component to component

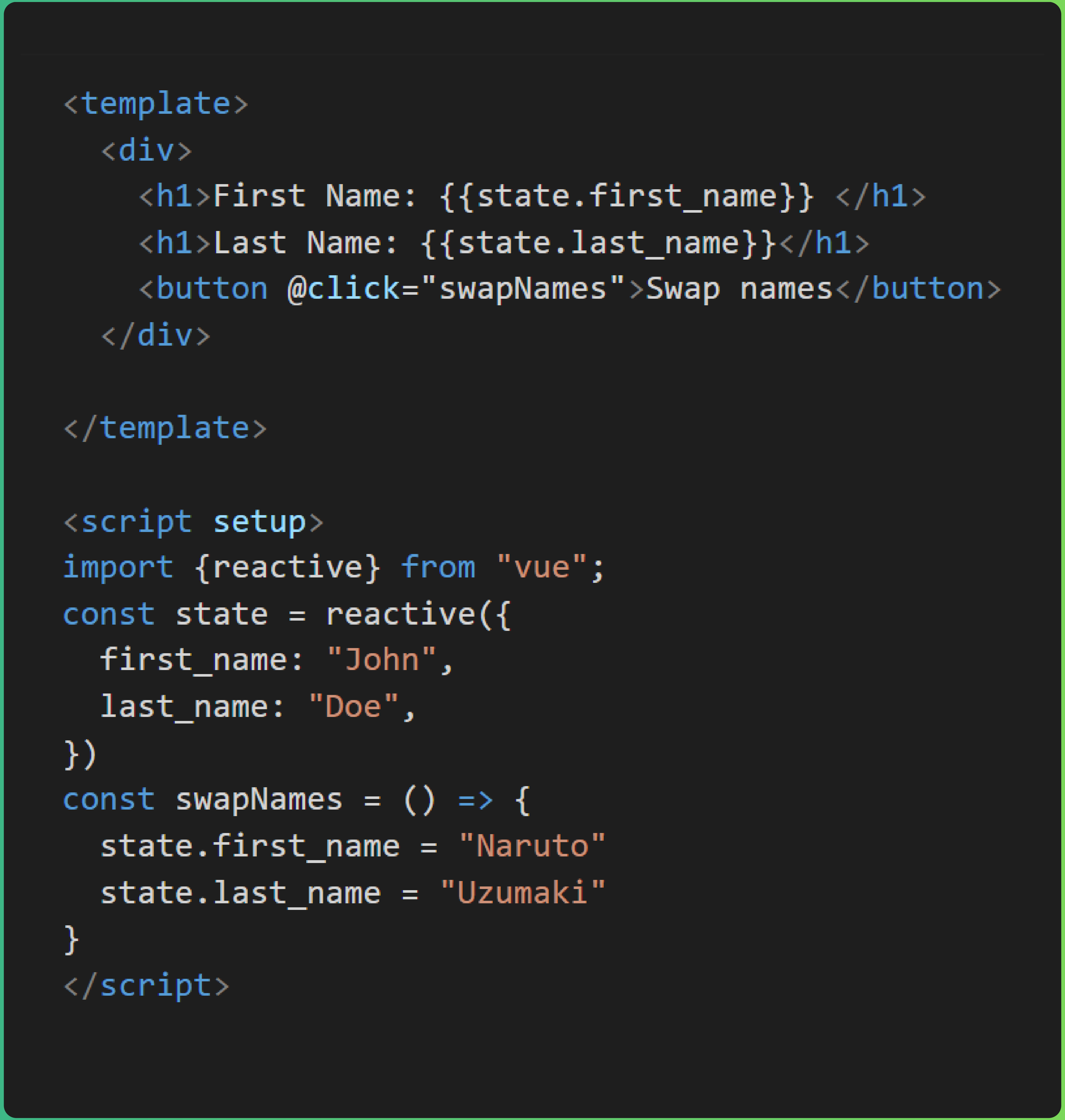
```
<template>
  <Hero
      title="Lorem Ipsum"
      description="Lorem Ipsum is simply dummy text"
  />
</template>

<script setup>
import Hero from "@/component/Hero.vue";
</script>
```

Hero.vue

```
<template>
<h1>{{title}}</h1>
<p>{{description}}</p>
</template>

<script setup>
const props = defineProps({
  title: String,
  description: String,
})
</script>

```

REACTIVE( ) TO DECLARE STATE

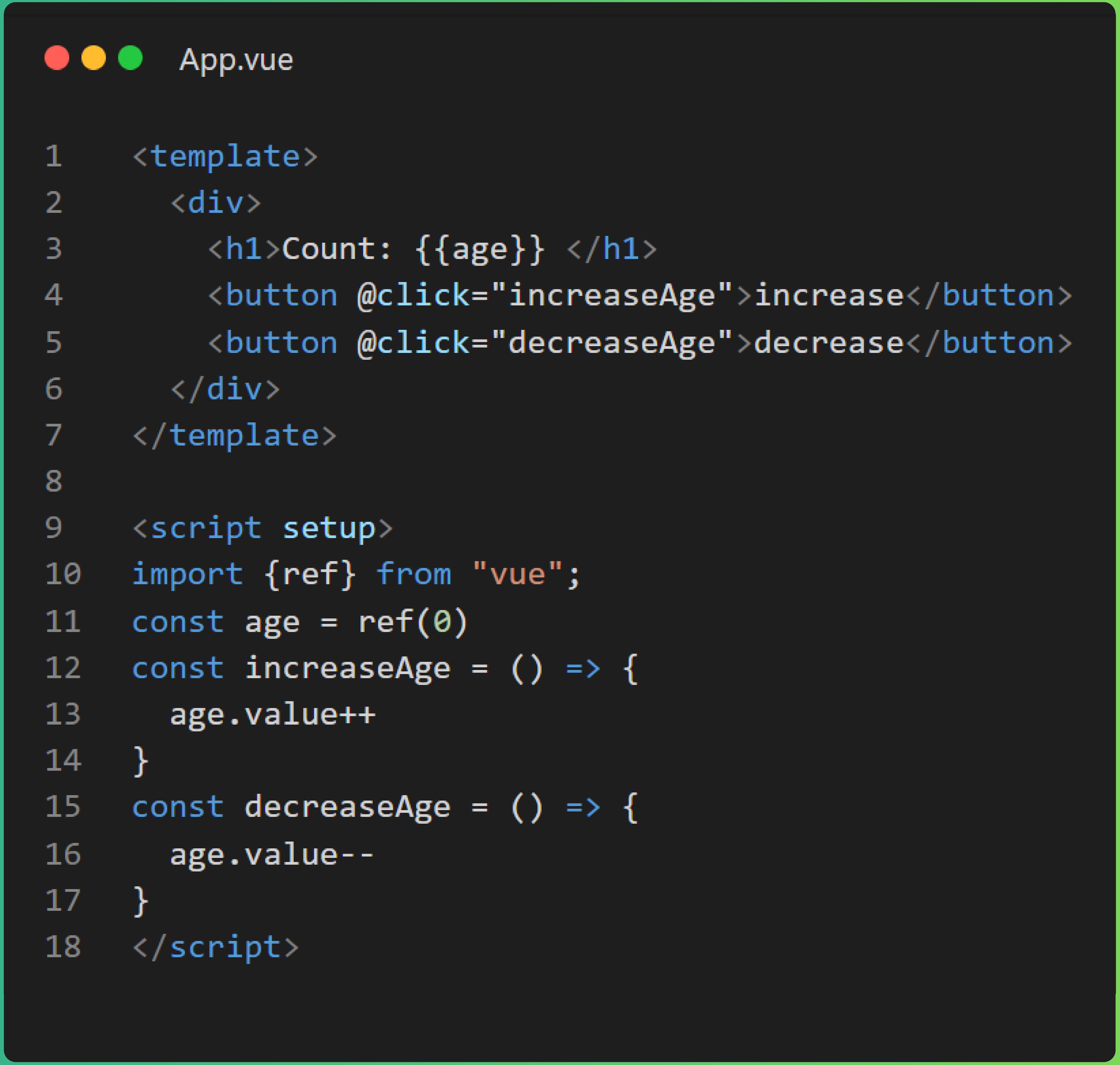
- Reactivity refers to the application's ability to update its user interface automatically when it's underlying data changed.
- The reactive() function is a powerful tool for creating reactive components.
- Reactive() function can only work with objects. Can't use it with primitives like strings or numbers
- To handle this limitation, Vue provides a second function for declaring reactive state in applications, ref()



```
<template>
  <div>
    <h1>First Name: {{state.first_name}} </h1>
    <h1>Last Name: {{state.last_name}}</h1>
    <button @click="swapNames">Swap names</button>
  </div>

</template>

<script setup>
import {reactive} from "vue";
const state = reactive({
  first_name: "John",
  last_name: "Doe",
})
const swapNames = () => {
  state.first_name = "Naruto"
  state.last_name = "Uzumaki"
}
</script>
```

REF( ) TO DECLARE STATE

- The ref() function can hold any value type, including primitives and objects. Therefore, we use it in a similar way to the reactive() function



```
<template>
  <div>
    <h1>Count: {{age}} </h1>
    <button @click="increaseAge">increase</button>
    <button @click="decreaseAge">decrease</button>
  </div>
</template>

<script setup>
import {ref} from "vue";
const age = ref(0)
const increaseAge = () => {
  age.value++
}
const decreaseAge = () => {
  age.value--
}
</script>
```

REF() VS REACTIVE() WHICH SHOULD YOU USE

- The significant difference between ref() and reactive() is that the ref() function allows us to declare reactive state for primitives and objects, while reactive() only declares reactive state for objects.
- Therefore, in real-world scenarios, you'll find more Vue code that uses ref() than reactive()

THE DOWNSIDES OF REF():

- It can be inconvenient to always have to use .value to access your state.
- You might not know if it has been initialized, and calling .value on null could throw a runtime error.

THE DOWNSIDES OF REACTIVE():

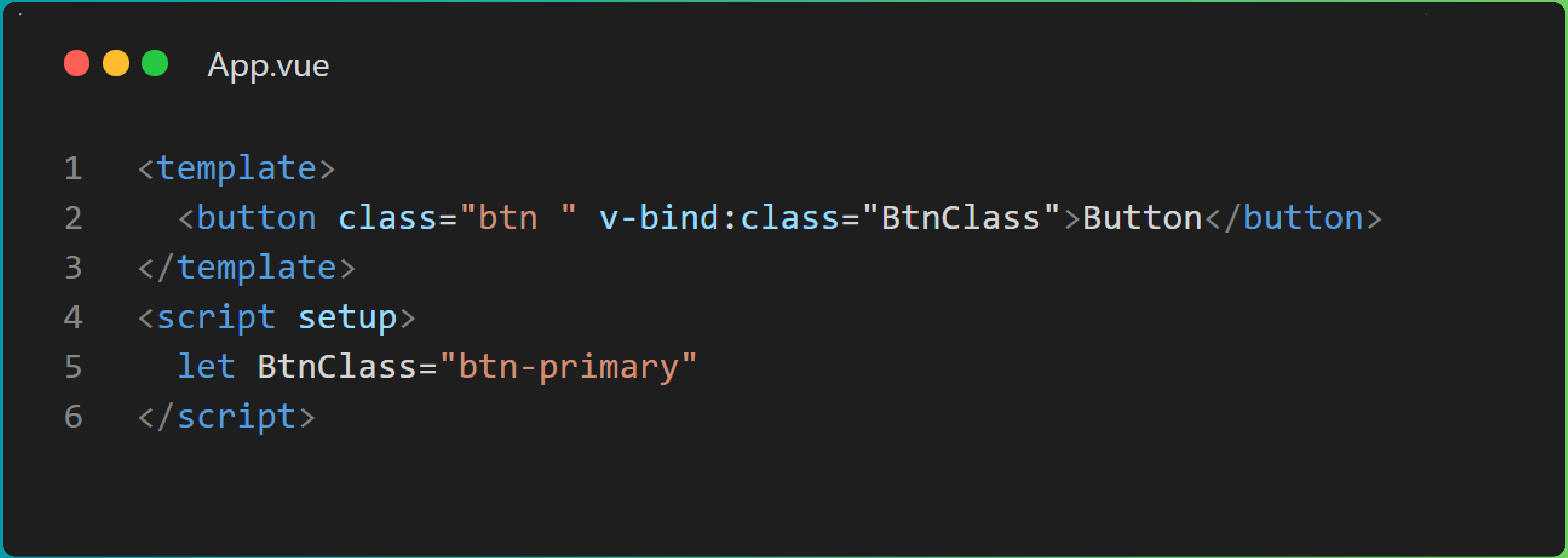
- Using reactive is that it cannot be used on primitives.
- You can only use reactive() on objects and object-based types like Map and Set

MIXING REF() AND REACTIVE()

- There's no rule or convention against mixing ref() and reactive()
- they can be used together without any technical drawbacks

CLASS AND STYLE BINDINGS

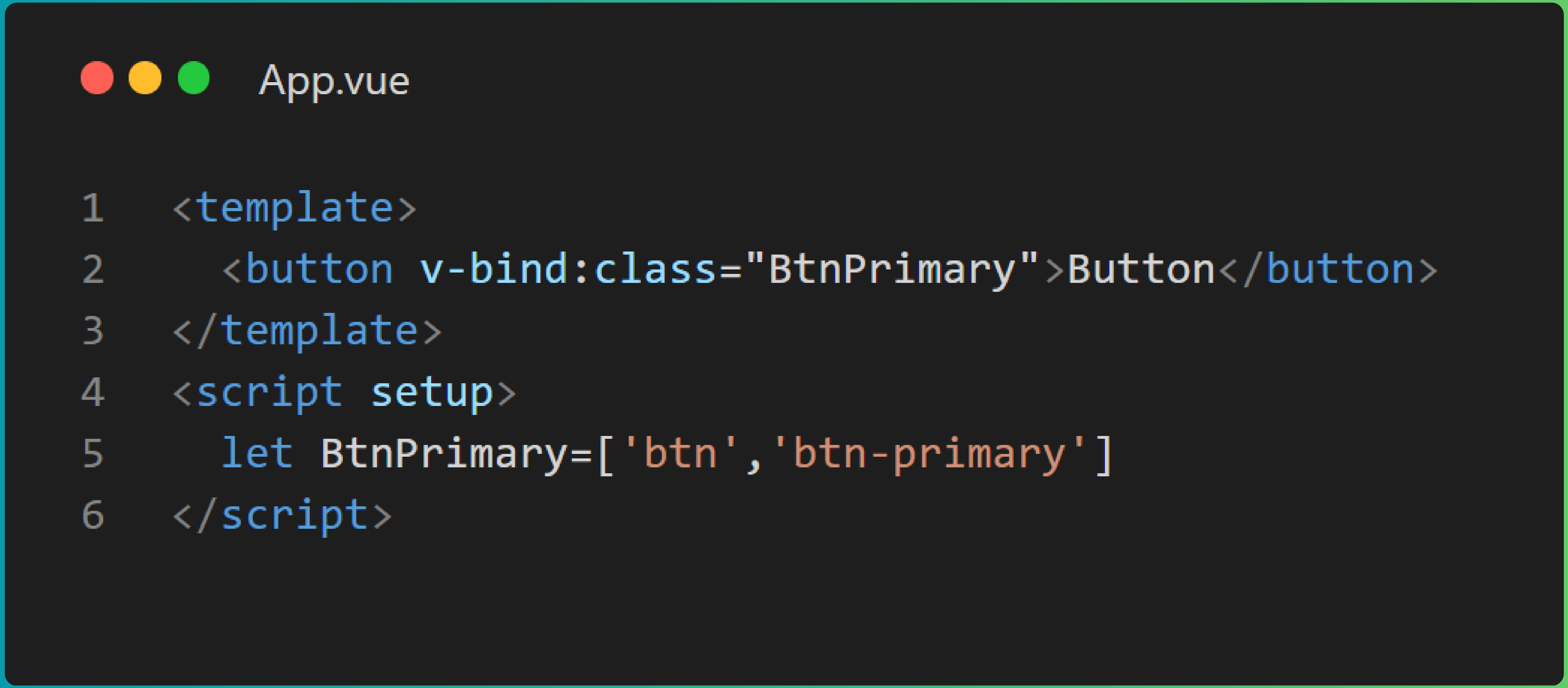
use v-bind


App.vue

```
<template>
  <button class="btn " v-bind:class="BtnClass">Button</button>
</template>
<script setup>
  let BtnClass="btn-primary"
</script>
```

CLASS AND STYLE BINDINGS

use v-bind

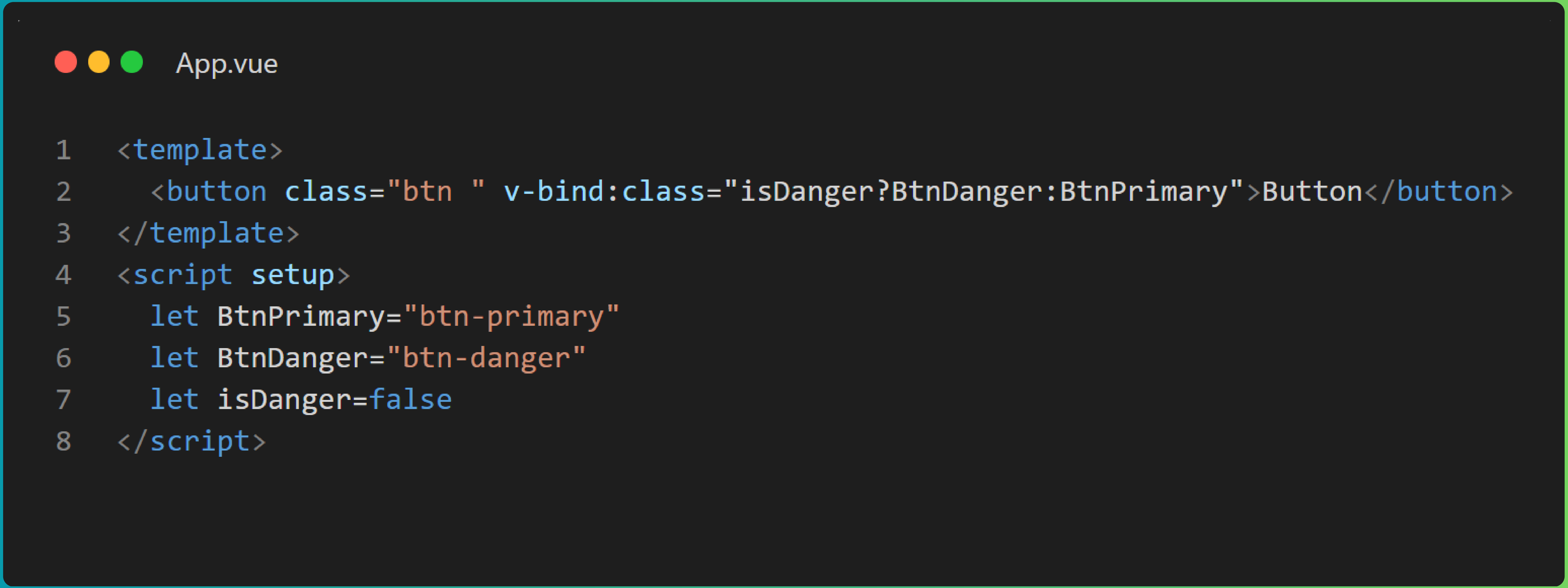


App.vue

```
<template>
  <button v-bind:class="BtnPrimary">Button</button>
</template>
<script setup>
  let BtnPrimary=['btn','btn-primary']
</script>
```

CLASS AND STYLE BINDINGS

use v-bind



App.vue

```
<template>
  <button class="btn " v-bind:class="isDanger?BtnDanger:BtnPrimary">Button</button>
</template>
<script setup>
  let BtnPrimary="btn-primary"
  let BtnDanger="btn-danger"
  let isDanger=false
</script>
```

WHY LARAVEL WITH VUE

Easy integration

Vue.js seamlessly integrates with Laravel, allowing developers to easily build dynamic user interfaces.

Reactive components

Vue.js uses reactive components, which means that the UI is automatically updated when the data changes.

Lightweight

Vue.js is a lightweight framework, which means that it doesn't add a lot of overhead to your application.

INTIGRATION

Integration Via Laravel Intertia

Vue.js seamlessly integrates with Laravel, allowing developers to easily build dynamic user interfaces.